ules were implemented on Sun Sparcstation and Silicon Graphics Iris hardware; the UI was implemented on the Iris using the GL graphics library. System 1.0 is scheduled for release within NAS by the end of 1992. This version of HNMS is written entirely in C and is portable to any UNIX system with a Berkeley sockets library and ISODE 7.0. The protocol definition and MIB are available from the authors. The user interface module for System 1.0 uses the X Window System and the Motif tool kit for all displays. All modules in this release are developed on the Sparcstation and Iris platforms.

7.0 CONCLUSION

Our intent is to have shown that there are definite performance and administrative benefits inherent in a distributed network management architecture. As we near the completion of development for this release, we can state that this is a successful implementation of a first generation distributed NMS. This case study should provide information useful to others who are interested in developing such systems. The development of a successor to SNMP is in progress. To date, what information has been released still does not describe an approach to distributed management of the type we are proposing [7][8]. We propose an object-based structure (for devices, not just variables), multi-administrator management, and efficient ways of cataloging and displaying data for a hierarchy of network elements. We hope that the development of this system will spur interest in the Internet community toward specifying a powerful, yet simple, standard for distributed network management and practical graphical representations for large networks.

8.0 REFERENCES

[1] Case, J., Fedor, M., Schoffstall, M., and Davin, J., “Simple Network Management Protocol”, RFC 1157, SNMP Research, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[2] McCabe, J., Schlecht, L., George, J., “NGMS Project Plan”, Technical Report RND-91-004, NAS Systems Division, NASA Ames Research Center, Moffett Field, CA, April 1991.

[3] Rose, M., and McCloghrie, K., “Structure and Identification of Management Information for TCP/IP-based Internets”, RFC 1155, Performance Systems International, Hughes LAN Systems, May 1990.

[4] McCloghrie, K., and Rose, M., “Management Information Base for Network Management of TCP/IP-based internets: MIB-II”, RFC 1213, Hughes LAN Systems, Performance Systems International, March 1991.

[5] Waldbusser, S., “Remote Network Monitoring Management Information Base”, RFC 1271, Carnegie Mellon University, November 1991.

[6] Rose, M., “The Simple Book: An Introduction to Management of TCP/IP-based Internets”, Prentice-Hall, Inc., New Jersey, 1991.

[7] Case, Jeffrey D., McCloghrie, K., Rose, M., and Waldbusser, S., “Protocol Operations for the Simple Management Protocol (SMP) Framework”, Internet Draft, University of Tennessee, Knoxville, Hughes LAN Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, July 1992.

[8] Case, Jeffrey D., McCloghrie, K., Rose, M., and Waldbusser, S., “Manager to Manager Management Information Base for the Simple Management Protocol (SMP) Framework”, Internet Draft, University of Tennessee, Knoxville, Hughes LAN Systems, Dover Beach Consulting, Inc., Carnegie Mellon University, July 1992.

needs information to be retrieved from the database, it will be done in one of two ways: 1) directly from a database client application, or 2) from an HNMS user interface module asking the server module for the data, which in turn will retrieve it from the database module and send it to the UI. To the HNMS server, the database module is just another client requesting or supplying information about the network. In practice, the server and database modules should never be located on the same host, as that would invalidate the database's use as a backup if the host were to go down.

Active Data. The database is used for two purposes by the HNMS. For the task of maintaining an up-to-date working set of information, the database maintains a set of tables containing records about HNMS objects. Each record is identified and indexed through its HNMS ID. These data must be maintained in a fail-safe location in case the HNMS server moves, or terminates unexpectedly, and needs to quickly recover its state. Active data can also be used for producing reports about the network, as it exists at that moment in time.

Inactive Data. There are also certain types of data which become aged; that is, they do not represent anything in the current state of the network. They are kept because they represent information that is useful in doing long-term traffic analysis, or simply for maintaining a record of events that have occurred in the network. There are two ways that data can become inactive. One way is to have an entire record become inactive. For example, a Processor may be removed from the network, or an Administrator may be retired. We cannot always remove these entries from the database because we may want to keep them for historical reasons. Inactive data are also created when we log specific fields from an active record. For example, an Interface object may have its ifInOctets recorded, along with a timestamp, every time it is reported to have changed (with a frequency up to its requested polling interval). In addition, every object will have a log of events associated with it. An event will consist of a timestamp, a status value, and a free-form text field. Events logged to an object can be generated by the HNMS (example: 5/5/92 05:01:00 4 “router ARC1 interface 3 down”), or by a human operator (example: 5/5/92 06:10:00 1 “There is a circuit problem on RFS62826, and I've called PSCN. --Bob.”). Each log entry will be related to an object that is currently active.

All object data, whether active or inactive, consist of sets of timestamped MIB variables. A user of the HNMS may call up data for an active Interface record and some subset of its past (inactive) event log entries. Although it seems more likely that a network administrator will make such a query directly to the SQL database, situations may arise where it will be necessary to go through a UI module (and thus the server) for the information.

5.0 ADDITIONAL MODULE TYPES

One of the advantages of a modular distributed system is that additional module types may be incorporated with relative ease. We are considering the development of a Rules-Based Intelligent Processor (RBIP) module. Once the HNMS system has been put to use at a sufficient number of sites, research into the automation of network management will be conducted. The RBIP is an expert system application which will adjust various HNMS parameters based on overall network traffic and usage of the system. The RBIP may also make short-term and long-term predictions about the network load and even make configuration changes in certain types of devices. We do not anticipate that the RBIP will require significant changes to the protocol. Rather, it will be similar to a UI module with a network administrator “built in.”

6.0 IMPLEMENTATION OF HNMS

Our prototype system was completed in April of 1991. We chose to implement this system in C and LISP to facilitate rapid prototyping of the protocol and internal data structures. The server and IO modare announced by the server; alternatively, the user could ask for the individual Subnets by name, and only those Subnets will be shown. This diagram is more detailed than the Site diagram and space is not a constraint. However, organization is important. The design of the Custom diagram must facilitate auto-generation while maintaining readability and avoiding obscurement. Our approach is a basic cartesian system with Subnets along the Y-axis, Processors along the X-axis, and Interfaces at intersection points. This method avoids the problem of crossed lines and provides a simple reference framework for the observer.

Any network element may be selected to create an *Object* diagram. This diagram is a textual display of the element's variables which is updated when the element changes. The Object diagram is the lowest level display in a hierarchical system of diagrams which allows the user to obtain more and more detail about a network and/or its elements.

In addition, HNMS offers several performance diagrams for visualizing network variables in troubleshooting and analysis. These real-time displays (updated at a regular interval), take the form of histograms for instantaneous single-element observation, line graphs for historical single-element observation, and an adaptation of the Site diagram for multi-element observation.

Since the UI may subscribe to any SNMP variable for any object, adding functionality to any of the diagrams is simply a matter of subscribing to different sets of MIB variables and providing graphical representations of them.

3.3 Color, Event Notification, and Customization

Color is an important visual cue in a network management system. In order to simplify diagrams and provide for a wider range of graphics workstations, we use only a few colors easily distinguishable from each other. An element's operational state is represented by four colors: *magenta* indicates Unknown, *green* indicates Responsive, *yellow* indicates Non-Responsive, and Unreachable is displayed as *red*. These states are determined by the server and are updated when they change. *White* is a neutral color used for displaying text and site boundaries and *orange* is used for routing paths.

In addition to operational state, HNMS displays the network administrative state: Normal (with no problems), Unresolved-Problem state, and Alarm state. Trouble ticket administration is carried out in conjunction with the administrative state designation, and operates in a distributed fashion. The system incorporates cross-references between the UI modules (through the server) such that network administrators at separate sites are able to participate in the tracking of problems spanning the entire network. The Alarm state, initiated by an operational state change to Unreachable, signals an operator of a problem. All UIs monitoring the questionable object will flash the object. An operator selects the object and requests the server to change the state to Unresolved Problem, flagging the object. The server validates the request and forwards the state change to the other UIs, thus informing other interested parties. When the problem is resolved, the operator again selects the element to request a change to Normal and removal of the flag, and the server informs the other UIs. At any time the operator or other users may open the current trouble log on the element to check progress or add comments. The log entries, like all other object data, consist of timestamped HNMS MIB variables and are maintained by the server and disseminated via subscriptions.

4.0 DATABASE

The database module is currently being developed to allow use of an SQL (Structured Query Language) database engine within the HNMS. Because of the large amounts of data generated by the HNMS and by the real-time nature of network management, it is essential that the database be able to complete transactions quickly. The database module itself is little more than a translator to convert HNMP messages to SQL commands. Such a module can be quickly written for any database engine. When a user

SNMP session, since the frequency of subscription replies would match the frequency of polling by the IO modules.

When setting up the HNMS architecture, care must be taken to choose intelligent locations for the IO modules, especially on a wide-area network. Optimally, SNMP polling is restricted to the local network at each site, with HNMP traffic traveling between modules over the WAN itself. The way to accomplish this is to situate the IO module on the boundary between the local network and the wide-area link. Many sites have a special LAN segment at this junction called the *isolation LAN*. We currently have an IO module running on a workstation situated on the isolation LAN at each of five AEROnet hub sites. Thus, no SNMP polling traffic is required to travel over the backbone.

A quantitative analysis of the performance benefits of a distributed NMS as opposed to a monolithic NMS is outside of the scope of this paper, but a document detailing such an analysis is available from the authors.

3.0 USER INTERFACE

3.1 Requirements

The most important requirements for the HNMS user interface include the following:

- *Provide automatically generated, accurate displays*: This is fundamental in a large, constantly changing network environment. With minimal instruction from the user, the UI should be able to automatically convert information from the server into a readable display. For accuracy, all of the elements in a diagram should be represented individually, not iconified as a group, which makes assumptions about an element's status and can be misleading. In addition, within the limits of the graphics hardware, displayed elements should not overlap or obscure each other.
- *Provide the observer with an immediate perception of the state of a large network without sacrificing readability:* The observer should be able to determine the general health of the network in a glance.
- *Provide notification of important events to the user.* The UI should offer an effective means of informing a network administrator that a problem has occurred.
- *Provide detailed information about a specific network element.* Provide access to the SNMP MIB variables.

3.2 Diagrams

HNMS provides four types of status diagrams, each representing the state of a network element by color. These diagrams are updated from the server as element status values change. When a new element is discovered, it is added to the diagram. Aside from object state, other useful information such as routing paths and response times may be displayed.

The *WAN* diagram depicts the state of a network and its routers over a geographical reference (e.g. a map of the US). Routers in the same site are clustered together and special consideration is taken to avoid obscuring one router with another. The routers are represented by dots and the links between them by lines.

The *Site* diagram can be created by selecting a site on the WAN diagram. This diagram represents all LANs that are connected to the routers at a given site. We determined that the interface is the primary informational element in this case, and all interfaces are displayed. The LANs are shown as concentric rings with their interfaces represented as dots, and important hosts and routers are placed around the periphery. This polar form makes a compact, centralized, representation for the observer.

The *Custom* diagram allows the user to construct a diagram with any set of network elements they wish to observe. When a Network is requested, individual Subnets within that network will appear when they

A second discovery method is a more directed version of the above. If an IO module is assigned a management domain which specifies a single interface IP address and SNMP community name, the IO module takes the initiative and immediately polls the address for a processor's definitive variables. This method is used for all of the wide-area routers in the NAS network, as they have separate community names.

A third discovery method utilizes the "next hop" entries in the routing table of each processor to walk through the network. By default, this is used only for routers, as determined by the ipForwarding variable.

The last discovery method simply consists of the reception of a trap from a previously unknown entity.

Regular polling for determining operational state is done using the sysUpTime and ifOperStatus variables. The polling used by the IO module is asynchronous; the module does not wait a given time, e.g. 4 seconds, for a reply. Rather, it sends out the request PDU and simply updates a timestamp for the object when a response is received. Once per second, the module checks the timestamps of all objects in its memory; if a timestamp is older than the time period set for a particular state change, it updates the operational state from Responsive to Non-Responsive, or from Non-Responsive to Unreachable, and informs the server of the fact. A network object may also change state if the IO module is directly informed that there is a state change; this may happen if a processor returns an ifOperStatus of down for one of its interfaces. In this way, an interface may change from Responsive to Unreachable directly, without going through the Non-Responsive state.

Special polling schemes may be used for certain managed entities. In particular, plans are being made to support the RMON MIB. While RMON agents are still limited by the current version of the SNMP protocol, their usefulness increases considerably when incorporated into a complete network management system which allows data gathering abstractions such as subscriptions.

2.7 Management Traffic

Most traffic between IO modules and the server, as well as between the server and UI modules, consists of replies to SubscribeData messages. Since a subscription is filled only when the value of a variable changes or when the status of an object changes, the inter-module HNMP traffic in a steady state is typically very low compared to the SNMP polling traffic between IO modules and their managed agents. If we compare this system to a conventional SNMP NMS, which continuously polls all managed agents from one point, it can be seen that the sum of SNMP traffic generated by each scheme is the same. However, the benefits of the HNMS architecture are apparent when the system is used at multiple locations. The HNMS makes it feasible to monitor a network that has multiple managerial points of control. The NAS wide-area network, AEROnet, connects six NASA centers (including NAS) and 34 other sites. The HNMS architecture provides the capability for the network to be monitored from any remote location. Under a typical network management architecture, each site would run a complete NMS. Each of these systems would contact every single network device through SNMP in order to build a representation of the network. Not only will this result in slightly different network representations at each station, it will generate an inordinate amount of network management traffic. The total number of polling sessions is the number of management stations times the number of managed objects. If there are 300 devices on the network, there would be 12000 SNMP polls going on during the time interval of one polling period. Under the HNMS scheme, there would be only 300 SNMP sessions and a few dozen HNMP connections, each of which is very lightweight compared to a continuous SNMP polling session since only changes are sent. These are obviously extreme examples, since not every site will have a desire or a need to monitor the entire network; they may subscribe to a subset of the information, or none at all. However, it is apparent that the distributed architecture will provide a savings in bandwidth in an extended network. Even under situations where the network is in a state of fluctuation and the status values of objects are changing rapidly, each HNMP connection is no more of a burden than an

In any subscription request, the timestamp field in each variable binding specifies the requested "currentness" of the data (e.g. 20 seconds). For each variable on every HNMS object, the server keeps track of the intervals requested by all UI modules. The lowest interval value is used as the actual polling interval by the IO module for that variable. Note that some variables are not gathered via SNMP but are created within the HNMS itself; for these variables, the interval field is left unused. In the replies, the timestamp field refers to the actual time that the variable held that value, as determined by the polling time. This is expressed in a standard time format.

In addition to the subscription method, a UI may ask for a set of variables to be sent to it just once. This is accomplished by means of the *GetData*, *GetNextData*, and *GetWalkData* messages. The GetData and GetNextData messages are similar to the SNMP Get and GetNext messages. The GetWalkData message asks for a complete walk of a branch of the MIB starting at each variable specified in the message, and is very effective for saving bandwidth. In typical SNMP management systems, a walk of a MIB branch requires the SNMP client to send a separate message for each requested variable, using the powerful GetNext operator [6]. In return, the client will receive a separate message containing the value of each variable. The GetWalkData message allows the request and the response to each be encoded in one message. When an IO module receives a walk request from the server, it performs the sequence of GetNext operations required to retrieve the entire branch. Since the IO module is typically situated on the same subnet as the entities within its management domain, the heavy SNMP polling of each set of elements is restricted to their immediate area and does not travel over the entire network.

The last message type dealing with object data is the *SetData* message. This message is a request by the UI to set the values of variables associated with an object, and requires the use of the key in a fashion similar to the SetParameters message. The server sets the appropriate values and forwards a SetData containing the new values to all modules subscribed to those variables. If the server denies the request, it sends an explanation to the UI module. In the case of SNMP variables associated with a processor, the server forwards the SetData message to the appropriate IO module, which in turn sends an SNMP Set message to the processor. The IO module then sends the server a SendData with the new value to confirm that it has been changed. Once the server updates its own object data, all subscribing modules are sent the new values.

There are three other messages which deal with entire HNMS objects, not with specific object data: *DeleteObject*, *DeactivateObject*, and *ActivateObject*. When an object is announced by the server, it is automatically assumed to be an active object. However, situations arise where we want to make the object inactive, meaning that it no longer represents an element currently on the network. In other instances, we may want to remove an object altogether from the server's memory. The DeactivateObject and DeleteObject messages are provided for this purpose. The final message, ActivateObject, is only provided for the sake of orthogonality. It is very rare that an object, once made inactive, will be re-activated.

2.6 Device Discovery and Data Collection

The primary discovery method used by the IO module is a three-stage process. Each IO module listens to traffic on its local Ethernet to learn Ethernet and IP addresses of other hosts. If it finds a new IP/Ethernet address pair which falls within one of its management domains, it enters the addresses into a cache and attempts to contact the source device via SNMP. The definitive SNMP variables for the processor (e.g. sysName, ifNumber) are then requested. If the device responds, the IO module then asks it for detailed information about all of its interfaces by simultaneously retrieving selected variables from the ifTable and walking the ipAddrTable. The variables are used as the input to a set of rules for determining whether the processor and its interfaces are unique objects not currently known by the IO module. If the processor has not been previously detected, the IO module announces it to the server. If the server already knows of the object, it sends its existing object data to the IO module; otherwise it assigns the new object an HNMS id and incorporates it into the HNMS system.

138.178.92.10, and a database module at 129.200.33.4. This is typically done by the network administrator who connects the first UI module after starting the server. The UI module gets the key, sends an AddDomains message, and returns the key.

Each IO module, in turn, is assigned a set of management domains by the server, based on the proximity of that IO module to the managed elements. In this case, the domain corresponds to a set of IP networks, subnets, and individual interfaces for which that IO module is directly responsible for monitoring. It is up to the server to determine which modules will monitor which domains. Currently, the algorithm is simply to assign all of the domains to the first IO module that connects. Subsequent modules receive subsets of the original set of management domains based on their location in the network, as determined by the IP addresses of their hosts. In most cases there is a one-to-one correspondence between IP subnets and IO modules.

A UI may request the server to stop managing a set of domains with a *DropDomains* message. This message is also used between the server and IO modules.

2.5 Data Exchange

Processor and Interface objects are first generated by the IO modules when they are discovered. These objects are given temporary HNMS id's and are sent to the server via an *Announce* message. An Announce message always includes the HNMS object and a set of object data. The variables used in an announcement are known as definitive variables, since they define the object; they represent the minimal set of variables required to distinguish an object from other objects of its class. Examples of definitive variables for an Interface object are its IP address and its layer 2 (e.g. Ethernet) address. Other variables are known as descriptive variables, which provide incidental information. Examples of these are the SNMP variables ifInOctets and ifOutOctets.

Once the server receives an announcement about an object, it checks the object's definitive variables against those of all of the objects it currently knows about. If the object appears to be unique, the server assigns it a permanent HNMS id and allows the object to be released to other modules. It immediately sends the object back to the announcing IO module, which replaces its temporary object with the one created by the server. From that point onward, the server receives data about the object by means of a *subscription* through the use of a *SubscribeData* message. A subscription consists of an HNMS id and a list of variables which the server would like the IO module to fill in. Each variable binding consists of a name:value pair and a timestamp. The HNMS objects and their object data are sent back to the server in *SendData* messages. A subscription is filled only when the value of a subscribed variable changes. Furthermore, only the changed variables are reported back to the subscribing module. If the server subscribes to 30 variables concerning an object and three of them change at any given time, those three will be sent back.

IO modules send announcements about Interface, Processor, Lan, and Link objects. Site, Administrator, PostalAddress, and Equipment objects are created by network administrators at UI modules and are announced to the server in a similar fashion. The server, being the final authority on object creation, does not allow duplicate objects to be created, from either the same module or separate modules. The rules for determining duplicates vary with each type of object, but all rely on the definitive variables in the object announcements. Finally, the server creates certain objects on its own, based on information it has gathered about IP addresses. The Internet, Network, and Subnet objects are created in this fashion.

Once an object has been created, it is announced to all UI modules. UI modules also retrieve information by means of subscriptions. The UI will choose some subset of the set of announced objects based on what the user wishes to monitor, and subscribe to certain variables regarding those objects. The UI may cancel variables in a subscription, or the entire subscription, with an UnsubscribeData message.

The next message that deals with connection management is the *Ack* message. HNMP requires that its messages be delivered reliably. Since the protocol is built upon UDP, it has provisions for lightweight connections between modules, including a sliding window retransmission scheme. The Ack message is used to provide an acknowledgement of each message that is received by a destination module in an HNMP connection, as well as to verify that a connection is still up when a module is deemed to have been silent for too long.

A UI module sends a *Goodbye* message to the server when it wishes to disconnect. Once the Goodbye has been acknowledged, the module may consider itself unconnected.

2.3 System Management

The HNMS has associated with it a number of operational parameters that can be set by users through a UI module. These parameters concern polling intervals, timeout intervals for determining state changes, intervals and buffer sizes associated with HNMP, and other aspects of module control. Some parameters are module-specific, while others are system-wide; the latter are maintained by the server although copies are kept within each module. If a user chooses to set a parameter on another module, or parameters concerning the entire system, a *SetParameters* message is employed. Each parameter is a triple of a parameter name, its value, and an integer defining the scope. The scope can identify a certain module or the entire system. If a module wishes to get the current value of a set of parameters, it uses a *GetParameters* message, which is similarly defined.

Problems may result if multiple UIs request that a certain parameter be set to conflicting values. For this reason, the notion of a *key* is employed, to ensure that only one UI at a time may be modifying a parameter. The key is a fictitious object which is maintained by the server. When a UI module wishes to send a SetParameters, it must first obtain the key by use of a *GetKey* message. If the key is not currently in use, the server responds with a *SendKey* message. At this point, the UI module has free reign to change as many system parameters as it desires. When it is finished, it uses a SendKey to return the key to the server. All key requests and parameter changes are logged by the server. Additional authentication measures may be employed; however the design of such measures are outside the scope of this paper. For the initial release of the HNMS, we have settled on a scheme in which only certain UI modules have the ability to use the key; these modules are used by select members of the network administration staff.

When the server refuses a request or sends informational messages to another module, it uses a *SendExplanation* message. This message is a combination of reference fields for internal use and a descriptive text block. The text block is a printable message which the UI may display on screen, at its discretion.

2.4 Management Domains

Before a server can begin its duties as a dissemination point for network data, it must be told which portions of the Internet it will be responsible for managing, and where it may find IO and database modules to enable the system to gather and store data. To facilitate this, the HNMS employs the concept of a *management domain*. A management domain is a quadruple comprising an IP address, an address mask, an SNMP community name, and a class, where class is one of the following: Internet, Network, Subnet, Interface, IO Module, Database Module. The first four class values are used for representing some entity that will be managed; in this case, the SNMP community name is used by the IO module which is eventually assigned the task of monitoring that section of the network via SNMP. The last two are used to inform the server of IP addresses where it can expect to find a module of the appropriate type.

A UI module sends the server a sequence of management domains by means of an *AddDomains* message. For example, the server may be given a list requiring it to monitor the network 138.178.0.0, the subnets 129.200.23.0 and 129.200.48.0, as well as the individual interfaces 135.102.10.1 and 135.102.10.2. In addition, the list may tell it to find IO modules at the addresses 129.200.23.1 and

Network	Internet, Administrator
Subnet	Network
Lan	Administrator
Link	Administrator
Processor	Site, Administrator
Equipment	Site, Administrator
Interface	Processor, Subnet, Lan, Link

Objects of class Internet and PostalAddress do not have references to any other objects.

All inter-module communication is done via a new Hierarchical Network Management Protocol (HNMP). HNMP, like SNMP, is built on top of UDP/IP, and ostensibly falls at layer 7 in the OSI network model. The protocol is specified in the Abstract Syntax Notation 1 (ASN.1) using its Basic Encoding Rules (BER); data types exchanged between the modules conform to the same subset of ASN.1 data types used by SNMP, as defined in the Structure of Management of Information (SMI) [3]. The protocol requires the use of a new HNMS MIB, which defines a set of variables (in addition to the standard SNMP variables [4]) in which object data are represented. In addition to transporting object data, HNMP facilitates a higher layer of network management with paradigms such as management domains and subscriptions, explained below. These allow a given module to receive a select set of management information about an arbitrary group of objects without repeatedly asking for it. More importantly, the protocol allows the system to uniquely identify data about any given HNMS object at any point in time, for the lifetime of the system, as each variable binding has a timestamp associated with it. We deemed that this level of flexibility was not possible with SNMP, even with solutions such as RMON (remote monitoring) devices [5] and other proxy agents [6], because of some fundamental differences in high-level versus low-level management. SNMP, by itself, provides only two levels of hierarchy: server (device agent) and client (management station). The HNMS, in conjunction with SNMP, uses four levels: device agent, IO module, server module, and UI or database module. Managing without such a scheme may not be difficult for a small network, but it is not practical for a network of the scope of the combined set of NAS networks.

2.2 Establishing Connections

The server and IO modules run as daemons on separate machines. In reality, the server is a special case of an IO module; at start-up, one IO module is given a flag telling it to be the default server.

Each server is distinguished by an HNMS community name. This is not to be confused with an SNMP community name, although the motivations behind both are similar; they are used to define an administrative classification of the managed devices and to provide a trivial form of authentication. The HNMS community name identifies the server and the object data which it disseminates. IO and database modules also identify themselves with an HNMS community name; either module type may only connect to a server whose community name matches its own. The community name is the only configuration that is given directly to any of these modules.

The UI modules initiate their connections to the server by sending an HNMP *Hello* message. The server responds with a *Welcome* message, assigning the module a module-id; this is an integer which is unique to that module, for any instantiation of an HNMS server. At start-up, a UI module presents the user with a pre-configured list of HNMS IO module hosts, one of which is the current location of the server. If the user selects a host on which an IO module is running, the IO module will respond with a *Redirect* message informing the UI module of the true location of the server. We are planning for the server module to be relocatable. If the host on which the server runs should crash, a new server is selected from the IO modules. We are also considering extensions to the protocol to allow peer to peer exchange of information between IO modules, such that there would be no definite server. A UI module would then be able to connect to the nearest IO module instead of a particular one identified as being the server.

ule, the *database module*, is a place to log long-term traffic studies, as well as keep current network state data. If the server machine should crash or become unreachable, a new server is started on a different machine; the server will then load the network state from the database. If the database module should become unreachable, data are queued until the database is again available. A fully functional HNMS consists of a server, a database, at least one IO module, and at least one UI module.

At this point, it will be useful to define a few terms. We refer to *network elements* as the actual entities that comprise a network -- routers, hosts, serial links, LAN segments, and individual interfaces. *HNMS objects* are the representations of these elements, and other abstractions, within an HNMS system. Each object is identified by a unique integer, its *HNMS id*, which is assigned by the server. There are, in total, eleven classes of HNMS objects:

Internet	the Internet
Network	an IP network
Subnet	an IP subnet
Lan	a layer 2 medium, part of a local-area network
Link	a layer 2 medium, part of a wide-area network
Interface	a device with a layer 2 address, layer 3 address optional
Processor	a host, router, or bridge
Site	a location, usually associated with a particular organization
Equipment	miscellaneous communications equipment
Administrator	a human, typically a network operator or site contact
PostalAddress	a postal mailing address or demarcation point

In addition to its class, each object has a subclass, and certain objects have a status. The subclass serves to further identify the type of object. Possible subclasses for the Interface class include Ethernet, FDDI, and serial link. The status identifies the operational and administrative state of the element, and is used with all HNMS object classes except Site, Equipment, Administrator, and PostalAddress. Values for the operational state include Responsive, Non-Responsive, and Unreachable. The operational state is ultimately set by the server, depending on whether the element has been heard from within certain preset time periods. Values for the administrative state include Alarm, Unresolved Problem, and Normal. The use of administrative state is covered in more detail in the section explaining the user interface.

Object data are the collection of information which describe a particular HNMS object; they consist of variables represented as name:value pairs which provide information apart from the class, subclass, and status. Representation of object data requires, in addition to the standard SNMP MIB variables [4], the use of a new HNMS MIB. This MIB contains separate groups of variables dedicated to describing the eleven classes of network objects. It also contains a "log entry" group whose variables are used for reporting alarms as well as for storing free-form textual data about any object.

For example, the object data for an Interface may include its Ethernet address (if it's subclass is Ethernet), its IP address, and its netmask. Object data also include variables defining the relations of an object. A relation exists if a variable contains a reference to the HNMS id of another object. For example, the object data for an Interface contains variables referring to the Interface's parent Processor, parent Subnet, and parent Lan or Link. Although we do not have space to reproduce the MIB here, we can provide a concise relational mapping between object classes and the other object classes which they are allowed to reference:

An object of class:	*has relations of class:*
Internet	*
PostalAddress	*
Administrator	PostalAddress
Site	Internet, Administrator, PostalAddress

- *scale to thousands of managed network elements:* At the start of our project, NAS provided network access to over 500 hosts. It was our goal to provide a means of monitoring all of these systems at once, presenting us with a challenge in the areas of data collection and intuitive displays.
- *minimize required human input by using automation:* The dynamic nature of a network in a research and development facility creates a network management editing nightmare. If editing were required to create a management display at NAS, the display would always be inaccurate. This makes automatic discovery and display of network elements desirable.
- *do not succumb to the problems being monitored or have a single point of failure:* The management system should be available and responsive at the most critical times and not be subject to network problems.
- *do not cause additional network problems:* The system should not overload the network with management traffic.
- *provide logging of network events:* In order to generate reports for troubleshooting and trend analysis, the system must provide a method of logging network data into a database.

1.2 Existing Management Software

At the outset of this project (late 1990) we conducted a survey of a dozen existing network management software packages and examined them according to our criteria (details can be found in reference [2]). Most of the commercial products used SNMP for data collection and X for display. However, most did not provide automatic discovery or display of elements, instead providing a graphical editor for the operator to create and update diagrams. With this editor, the operator could introduce meaningless and confusing objects into the diagram reducing the tool's usefulness and authority. In addition, none of the products addressed the issue of scalability in data collection or display; that is, none of the products could have displayed the entire NAS network in a readable form. For these reasons we decided to develop HNMS.

1.3 Development Approach

The size and complexity of the NAS local and wide-area networks are the most important issues in management, creating problems in the areas of data collection and display, and network load. We have approached these problems through a distributed and hierarchical strategy. Since no standard currently exists for a true distributed network management architecture, our design builds on existing standards currently used in network management. With the aforementioned issues in mind, this paper presents an implementation of a distributed NMS and examines how architectural features of the system allow for better fault-tolerance, creation of less management traffic, and a more structured scheme for cataloging network elements, as opposed to that provided by a conventional NMS.

2.0 ARCHITECTURE

2.1 General

The system consists of four types of modules. The modules are software entities, typically residing on separate hosts throughout the network. The *server module* is the hub of the management system. It provides a center for dissemination of topology and status information. *User Interface (UI) modules* reside on graphical workstations and provide users with access to real-time or logged data. *Input/Output (IO) modules* reside on hosts located at strategic points within the local-area and wide-area networks and handle the actual data collection. IO modules use SNMP and layer 2 monitoring for data collection. The IO modules pass filtered management data up to the server. The server, in turn, sends relevant information to the UI modules participating in the management system. Data are sent only when their values have changed. Thus the hierarchical approach allows us to avoid flooding a network with management traffic and creating bottlenecks where management information is processed. The fourth type of mod-

The NAS Hierarchical Network Management System[1]

Jude A. George, Leslie E. Schlecht
jude@nas.nasa.gov, schlecht@nas.nasa.gov

Computer Sciences Corporation
M/S 258-6, NASA Ames Research Center, Moffett Field, CA 94035, USA

Abstract

This is a case study of the development of a network management system for the Numerical Aerodynamics Simulation facility at NASA Ames Research Center. Issues relevant to IP network management are identified and examined. Experience has shown that traditional monolithic network management systems cannot address all of these issues in a large network; in many cases, a distributed, hierarchical system is more suited to this task. With these issues in mind, a scalable architecture for monitoring large networks is presented: the Hierarchical Network Management System. Methods for gathering, storing, and transporting network information within this system are described. A user interface incorporating design principles for displaying and managing large networks is also presented. We expect that the presentation of this architecture will be of value to other members of the Internet community who would participate in the development of an open standard for distributed network management.

Keyword Codes: K.6.0; C.2.4; H.5.2

Keywords: Management of Computing and Information Systems, General; Distributed Systems; User Interfaces

1.0 INTRODUCTION

The Hierarchical Network Management System (HNMS) project is a result of the continuing installation of large, high speed local and wide-area networks for the Numerical Aerodynamic Simulation (NAS) facility at the NASA Ames Research Center. The complexity of these new networks has forced us to look into a single complete network management solution in lieu of separate management tools specific to each type of network hardware.

1.1 Requirements for NAS

There are a number of issues involved in the selection of a network management system. For NAS, we developed the following criteria:

- *use open published standards where available (SNMP and X Window System):* To facilitate the integration of new network hardware and reduce reliance on any one vendor, data collection should be carried out through a standard protocol. For availability on a wide range of platforms, the user interface should be based on a standard window system. SNMP was chosen for data collection and X for display because of their wide vendor support.
- *present a clear, precise representation of network elements:* The state of a network element must be defined meaningfully, and with no assumptions. Additionally, there must be no misleading or confusing information introduced through editing or group representation.

1. Integrated Network Management, III (C-12); H.-G. Hegering and Y. Yemini (Editors)
Elsevier Science Publishers B.V. (North-Holland). (c) 1993 by the authors.